This lecture talks about memory and the need of pointers.

Memory is a critical part of computers.

C D C 66 00 is considered as the first super computer and had up to 982 kilo bytes of memory. Kilo means a thousand. In 2019 dollars, this computer would cost about 19 million dollars. In contrast, an HTC 10 evo phone costs about 199 dollars and has 32 giga bytes of memory. Giga means a billion. Thus, the mobile phone has more than 32,000 times of memory and cost about 0.001% of a super computer. You can imagine the tremendous progress in technologies.

Let’s review the structure of memory in computers.

Computer memory is organized in address-value pairs. Each row has an address and a value is stored at that address. You can think of the addresses in computer memory as the addresses of houses. The values are the names of the families living in the houses. For example, along a street. The first house lives the family of Johnson. The second address belongs to the family of Smith. The third address is for the family of Taylor.

In the stack memory, the first column is the frame. Each function has its own frame. Consider the frame of f3 in this example. Function f3 is called by f1 at line 4. Thus, the return location is line 5. The value returned from f3 is written to aee so the value address is the address of "Ae". and it is 102. Please notice that the value address is stored in the stack memory and space is needed for storing this information. The value address is stored at address 104.

Do not mix the value and the address. The value address is 102 and the address is stored at location 104.

Let me introduce a new concept about the scope of variables.

A function can see only its frame.

For example, inside f3, the variable aee is not visible. Thus, if we want to assign 5 to aee, G C C will tell us this is an error. Function f3 cannot see the variable aee because aee resides in the frame of f1.

It is possible to use aee as the name of an argument or a local variable in f3. However, this aee is different from the aee in f1 because they occupy different memory addresses. The aee in f1 uses address 102. The aee in f3 uses address 105. Even though they have the same name, they mean different things. This is like two families have the same last name living in two different houses.

Earlier, I said the concept of frames and symbols are for humans. Computers do not know frames or symbols. Computers know only addresses and values. Thus, the same name aee does not mean anything to computers.

Function f3 cannot see aee in f1’s frame.

Each function can see its own frame only.

This will cause serious problems and make some problems very difficult to solve.

Let’s consider writing a swap function.

The swap function takes two input arguments. After the function call, the two arguments’ values are swapped.

This is an example. Suppose aee is 5 and bee is 7. After calling the swap function, aee should be 7 and bee should be 5.

Let’s consider the first attempt. This function takes two arguments and returns the second. The returned value is stored in aee.

This does not work because aee becomes 7 and bee is still 7.

Let’s consider another method. This function uses a local variable tee to store the value from argument aee. Then, the function assigns bee’s value to aee. The value of tee is assigned to bee.

Will this work?

Since this is a function call, let’s use the stack memory to understand how things work.

Suppose the program has just finished line 2. The value of aee is 5 and the value of bee is 7.

The program is ready to call the swap function.

At this moment, the stack memory is shown here.

The program calls the swap function and has finished the first line of the function. In this slide, the first line of the function is line cee.

The stack memory is shown here. The return location is line 4.

Since the swap function returns nothing, there is no value address.

The arguments aee and bee are 5 and 7 respectively.

The local variable tee is the value of aee so tee’s value is 5.

The program moves to the next line and assigns bee’s value to aee. Aee’s value becomes 7.

The next line assigns tee’s value to bee. Bee’s value becomes 5.

This is what the stack memory looks like before the function finishes and the program goes to line 4.

When the swap function finishes and the program does to line 4, the top frame of the stack memory is popped.

Do you notice that aee and bee are unchanged?

What is happening? Why?

.

Remember that the aee in swap function is not the same aee in line 1. They occupy different memory addresses.

To think of this in a different way.

We can rename aee and bee in the swap function to x and y. The swap function is exactly the same.

This is the reason I said earlier that computers do not care about symbols. Computers care about only addresses and values. Aee, bee, x, and y are symbols for human programmers. We can rename the symbols and the program is exactly the same.

The next lecture will explain how we can write a correct swap function by using pointers.